Amendments to the Specification:

Please replace the paragraph beginning at page 2, line 8 with the following amended paragraph:

One or more of the following features can be included. Resolving can include determining override conditions in rule collision events. Generating can include analyzing the rule set with a business logic generation utility optimized for one of a plurality of target programming languages and generating optimized business logic for the selected target programming language. The target programming language can be Java, C++, C#, Jython, JavaScript and ~~visual basic~~ Visual Basic. The business logic generation utility's generated processing logic can include a series of calls to a working memory database to retrieve, manipulate and update data.

Please replace the paragraph beginning at page 11, line 12 with the following amended paragraph:

FIG. 9 shows a basic architecture of inference engines. This architecture is driven by the dynamic conflict resolution requirement of expert systems. Each rule's premises are pattern matched against the known facts. All rules whose premises are fully matched are activated and placed on the agenda (conflict set). Activated rules are only fired one at a time using a conflict resolution algorithm determining which rule to fire next. When a rule fires, it may update the known facts. Before the next rule fires, another pattern match cycle occurs to make sure the agenda is up-to-date. This may also result in deactivating rules on the agenda whose premises are no longer satisfied. The most expensive part of this architecture, in terms of computer resources, is the pattern matching. It can take up to 90% of processing time. This is why there are numerous pattern matching algorithms (e.g. RETE and TREAT), each optimized for a specific problem domain. The pattern match-conflict resolution cycle shown in FIG. 9 allows rules to impact other ~~rules~~ rules' execution. For example, a rule can deactivate another rule by updating working memory so that the latter rules' premises are no longer satisfied.

Please replace the paragraph beginning at page 12, line 23 with the following amended paragraph:

The same dependency analysis utility can be used in conjunction with various business logic generation utilities, each optimized for a different target programming language. In FIG. 13, a rule engine where the business rules are captured in a declarative high-level language is shown. A code generator/compiler 502 converts rules 504 into low-level business logic 508 coded in and optimized for a programming language, such as, for example, Java, C++, Jython, or ~~JavaScrip~~ JavaScript. A base library 510 that implements the rule engine's working memory is included. This is analogous to an in-memory relational database. It provides many of the relational algebra operations available in a relational database such as union, intersect, difference, product, restrict, project, join, divide, and so forth. The business logic 508 generated is a series of calls to the working memory database to retrieve, manipulate, and update the data.

Please replace the paragraph beginning at page 13, line 5 with the following amended paragraph:

FIG. 15 includes example ~~Python~~ Jython code for the processing of the three business rules in FIG. 10.